Captone_6_BOW_TFIDF_W2V

March 7, 2024

[1]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

[2]:
```
import numpy as np
import pandas as pd
```

Loading data set

[3]:
```
df = pd.read_csv("/content/drive/MyDrive/Capstone Semester 6/
  ↪software_requirements_extended.csv")
```

[4]:
```
df.head()
```

[4]:
```
  Type                                        Requirement
0   PE  The system shall refresh the display every 60 …
1   LF  The application shall match the color of the s…
2   US   If projected  the data must be readable.  On …
3    A   The product shall be available during normal …
4   US   If projected  the data must be understandable…
```

[5]:
```
df["Requirement"][1]
```

[5]:
```
'The application shall match the color of the schema set forth by Department of
Homeland Security'
```

Count of Various Classes

[6]:
```
df["Type"].value_counts()
```

[6]:
```
FR     312
F      209
NFR    110
US      63
O       58
SE      56
PE      54
LF      34
A       21
```

```
SC      21
MN      17
L       10
FT      10
PO       2
Name: Type, dtype: int64
```

Check if NULL or DUPLICATE entries

[7]:

```
df.isnull().sum()
```

[7]:

```
Type           0
Requirement    0
dtype: int64
```

[8]:

```
df.duplicated().sum()
```

[8]:

```
0
```

1 Preprocessing raw data

1. Removing stopwords
2. lemmetization
3. Removing unwanted symbols
4. Lowercasing

[9]:

```
import nltk
nltk.download('stopwords')
nltk.download('wordnet')
import re
import string
import numpy as np
from nltk.corpus import stopwords
from nltk.tokenize import TweetTokenizer
from nltk.stem import WordNetLemmatizer
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data…
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data…
```

[10]:

```
stopwords_english = stopwords.words('english')
lemmatizer = WordNetLemmatizer()
tokens = TweetTokenizer(preserve_case=False, strip_handles=True,reduce_len=True)

def process_text(text):
  text = re.sub(r'\d', '',text)
  unwanted_symbols = ['€', ' ', 'â', '<','%']
  for symbol in unwanted_symbols:
    text = text.replace(symbol, '')

  text_tokens = tokens.tokenize(text)
  clean_text=""
  for word in text_tokens:
    if (word not in stopwords_english and  word not in string.punctuation):
      if len(word)<=2:
        continue
      lemma_word = lemmatizer.lemmatize(word)
      clean_text = clean_text + " " + lemma_word
  return clean_text.lower()
df['cleaned_text'] = df['Requirement'].apply(process_text)
```

[11]:
```
df.head()
```

[11]:
```
  Type                                         Requirement  \
0   PE  The system shall refresh the display every 60 …
1   LF  The application shall match the color of the s…
2   US   If projected  the data must be readable.  On …
3    A   The product shall be available during normal …
4   US   If projected  the data must be understandable…

                                        cleaned_text
0          system shall refresh display every second
1   application shall match color schema set fort…
2   projected data must readable projection scree…
3   product shall available normal business hour …
4   projected data must understandable projection…
```

[12]:
```
X = df.iloc[:,2]
y = df["Type"]
```

[13]:
```
X
```

[13]:
```
0              system shall refresh display every second
1       application shall match color schema set fort…
2       projected data must readable projection scree…
3       product shall available normal business hour …
4       projected data must understandable projection…
                             …
972               designated phone number user send text
973     text sent number sent api system reply user a…
974     question understood api system send text cont…
975     upon usb plugged system shall able deployed o…
976     system shall able handle customer logged conc…
Name: cleaned_text, Length: 977, dtype: object
```

[14]:
```
y
```

[14]:
```
0      PE
1      LF
2      US
3       A
4      US
       ..
972    FR
973    FR
974    FR
975    FR
976    FR
Name: Type, Length: 977, dtype: object
```

2 Label encoding the output class

[15]:
```
from sklearn.preprocessing import LabelEncoder

encoder = LabelEncoder()

y = encoder.fit_transform(y)
```

[16]:
```
y
```

[16]:
```
array([ 9,  5, 13,  0, 13, 12, 13,  9,  1,  1,  1,  4,  4,  4,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  8,  8,  8,  8,  8,  5,  5,  5, 10, 13, 13, 13,
       13, 13,  9,  9,  9,  9,  0,  0, 11, 11, 11,  8,  8,  8,  8,  8, 12,
       12, 12, 12, 12, 12, 12,  4,  4,  4,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  8,  8,  8,  8,  8,  5,
        5, 13, 13, 13, 13, 13, 13, 13, 13, 13,  9,  0,  3, 11, 11, 11,  8,
        8,  8,  8,  8,  8,  8, 13,  6,  6,  6, 13,  6, 13,  8,  8, 12, 12,
       12, 12,  3, 12, 11, 13,  8,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        9,  9,  8,  8, 12, 12,  4,  8,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9,
        5,  5,  5, 13, 13, 13, 13, 13, 13, 13, 13, 13,  9,  9,  9,  9, 12,
        0,  0,  0,  3,  3, 11, 11, 11, 11,  6,  8,  8,  8,  8,  8,  8,  8,
        6,  9,  9,  0, 13,  0,  8, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12,
       12, 12, 12, 12,  5,  5,  5,  8,  4,  4,  4,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  9,  9,  9,
        9,  8,  8,  0,  6,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  5,  5,  5,  5,  5,  5,  5,  0,  9,  9,  9, 13, 13,
       12,  9,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1, 13, 13,  9,  9,  9,  0,  0,  6, 12, 12, 12,
        5,  5,  5,  5, 13, 13, 13,  9,  9,  9,  9,  9,  0,  3,  3, 11,  8,
        8,  6,  6,  6, 12, 12, 12,  5,  5,  5,  5, 13, 13, 13, 13, 13, 13,
        0, 11, 11,  8,  8,  6,  6, 12, 12, 12, 12,  5,  5,  5, 13, 13, 13,
       13,  9,  0,  8,  9,  9,  8,  8,  8,  0,  5,  5, 13, 13, 13, 13,  9,
        9,  9,  9,  9,  9,  0,  0,  3,  3,  3,  3, 11, 11, 11, 13, 13, 10,
       12, 12, 12,  1,  1,  1,  1, 11, 11, 11, 11,  8,  8,  8,  1,  1,  1,
        1,  1, 13, 13, 13, 13, 13, 13, 13, 13,  9,  9,  0,  8,  8,  8,  8,
        8, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12,  1,  1,  1,  1,  1,  1,
        1,  6,  6,  1,  1,  1,  1,  1,  1,  1,  1,  1,  0,  8,  8,  8,  6,
        6, 12, 12,  5,  5, 13,  9,  1,  1,  1,  1,  2,  2,  7,  2,  7,  2,
        2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
        2,  2,  2,  2,  7,  7,  7,  7,  7,  7,  2,  2,  2,  2,  2,  2,  2,
        2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  7,
        7,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
        2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
        2,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,
        7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,
        7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,
        7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,
        7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,
        7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  2,
        2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
        2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
        2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
        2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
        2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
        2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
        2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
        2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
        2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
        2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
        2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
        2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
        2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
        2,  2,  2,  2,  2,  2,  2,  2])
```

3 Stratified Spliting of Data into train and test set

```
[17]: from sklearn.model_selection import train_test_split
      X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.
       ↪2,stratify=y,random_state=1)
```

```
[18]: X_train.shape
```

```
[18]: (781,)
```

Count of various classes in Train set

```
[19]: pd.DataFrame(y_train).value_counts().sort_index()
```

```
[19]: 0      17
      1     167
      2     249
      3       8
      4       8
      5      27
      6      14
      7      88
      8      46
      9      43
      10      2
      11     17
      12     45
      13     50
      dtype: int64
```

Count of various classes in Test set

```
[20]: pd.DataFrame(y_test).value_counts().sort_index()
```

```
[20]: 0      4
      1     42
      2     63
      3      2
      4      2
      5      7
      6      3
      7     22
      8     12
      9     11
      11     4
      12    11
      13    13
      dtype: int64
```

```
[21]: encoder.classes_
```

```
[21]: array(['A', 'F', 'FR', 'FT', 'L', 'LF', 'MN', 'NFR', 'O', 'PE', 'PO',
             'SC', 'SE', 'US'], dtype=object)
```

4 Applying *Bag-Of-Words* Text Vectorization Technique

```
[22]: from sklearn.feature_extraction.text import CountVectorizer
```

```
[23]: cv = CountVectorizer(max_features=1400)
```

```
[24]: X_train_bow = cv.fit_transform(X_train).toarray()
      X_test_bow = cv.transform(X_test).toarray()
```

Shape of text document:

```
[25]: X_train_bow.shape
```

```
[25]: (781, 1400)
```

Naive Bayes Classifier

```
[26]: from sklearn.naive_bayes import GaussianNB
      from sklearn.metrics import accuracy_score,confusion_matrix

      gnb = GaussianNB()

      gnb.fit(X_train_bow,y_train)

      y_pred = gnb.predict(X_test_bow)

      accuracy_score(y_test,y_pred)
```

```
[26]: 0.7091836734693877
```

Decision Tree Classifier

```
[27]: from sklearn.tree import DecisionTreeClassifier

      clf = DecisionTreeClassifier(random_state=42)

      clf.fit(X_train_bow, y_train)

      y_pred = clf.predict(X_test_bow)

      accuracy_score(y_test,y_pred)
```

```
[27]: 0.7091836734693877
```

Random Forest Classifier

```
[28]: from sklearn.ensemble import RandomForestClassifier
      rf = RandomForestClassifier(random_state=42)

      rf.fit(X_train_bow,y_train)
```

```
y_pred = rf.predict(X_test_bow)
accuracy_score(y_test,y_pred)
```

[28]:

```
0.75
```

SVM Classifier

[29]:

```
from sklearn.svm import SVC
from sklearn.metrics import classification_report

svm_classifier = SVC(kernel='linear', decision_function_shape='ovr')

svm_classifier.fit(X_train_bow, y_train)

y_pred = svm_classifier.predict(X_test_bow)

accuracy_score(y_test,y_pred)
```

[29]:

```
0.7755102040816326
```

KNN Classifier

[30]:

```
from sklearn.neighbors import KNeighborsClassifier

knn_classifier = KNeighborsClassifier(n_neighbors=8)

knn_classifier.fit(X_train_bow, y_train)

y_pred = knn_classifier.predict(X_test_bow)

accuracy_score(y_test,y_pred)
```

[30]:

```
0.6173469387755102
```

Xgboost Classifier

[31]:

```
import xgboost as xgb

dtrain = xgb.DMatrix(X_train_bow, label=y_train)
dtest = xgb.DMatrix(X_test_bow, label=y_test)

params = {
    'objective': 'multi:softmax',
    'num_class': 14,
    'eval_metric': 'mlogloss'
}

num_rounds = 60
xgb_model = xgb.train(params, dtrain, num_rounds)
```

```
y_pred = xgb_model.predict(dtest)

accuracy_score(y_test,y_pred)
```

[31]:
```
0.7755102040816326
```

5 Applying *TF-IDF* Text Vectorization Technique

[32]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
```

[33]:
```
tfidf = TfidfVectorizer()
```

[34]:
```
X_train_tfidf = tfidf.fit_transform(X_train).toarray()
X_test_tfidf = tfidf.transform(X_test)
```

Naive Bayes Classifier

[35]:
```
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score,confusion_matrix

gnb = GaussianNB()

gnb.fit(X_train_tfidf,y_train)

y_pred = gnb.predict(X_test_tfidf.toarray())

accuracy_score(y_test,y_pred)
```

[35]:
```
0.6836734693877551
```

Decision Tree Classifier

[36]:
```
from sklearn.tree import DecisionTreeClassifier

clf = DecisionTreeClassifier(random_state=42)

clf.fit(X_train_tfidf, y_train)

y_pred = clf.predict(X_test_tfidf)

accuracy_score(y_test,y_pred)
```

[36]:
```
0.6836734693877551
```

Random Forest Classifier

[37]:

```
rf = RandomForestClassifier(random_state=42)

rf.fit(X_train_tfidf,y_train)
y_pred = rf.predict(X_test_tfidf)

accuracy_score(y_test,y_pred)
```

[37]:

```
0.75
```

SVM Classifier

[38]:

```
from sklearn.svm import SVC
from sklearn.metrics import classification_report

svm_classifier = SVC(kernel='linear', decision_function_shape='ovr')

svm_classifier.fit(X_train_tfidf, y_train)

y_pred = svm_classifier.predict(X_test_tfidf.toarray())

accuracy_score(y_test,y_pred)
```

[38]:

```
0.7857142857142857
```

KNN Classifier

[39]:

```
from sklearn.neighbors import KNeighborsClassifier

knn_classifier = KNeighborsClassifier(n_neighbors=13)

knn_classifier.fit(X_train_tfidf, y_train)

y_pred = knn_classifier.predict(X_test_tfidf)

accuracy_score(y_test,y_pred)
```

[39]:

```
0.6887755102040817
```

Xgboost Classifier

[40]:

```
import xgboost as xgb

dtrain = xgb.DMatrix(X_train_tfidf, label=y_train)
dtest = xgb.DMatrix(X_test_tfidf, label=y_test)

params = {
    'objective': 'multi:softmax',
    'num_class': 14,
    'eval_metric': 'merror'
}

num_rounds = 100
xgb_model = xgb.train(params, dtrain, num_rounds)

y_pred = xgb_model.predict(dtest)

accuracy_score(y_test,y_pred)
```

[40]:

```
0.21428571428571427
```

6 Word2Vec Approach

[42]:

```
from gensim.models import Word2Vec
from keras.preprocessing.text import text_to_word_sequence
import numpy as np

# Tokenize sentences into words
X_train_tokenized = [text_to_word_sequence(sentence) for sentence in X_train]
X_test_tokenized = [text_to_word_sequence(sentence) for sentence in X_test]

# Train Word2Vec model
word2vec_model = Word2Vec(sentences=X_train_tokenized + X_test_tokenized,␣
 ↪vector_size=100, window=5, min_count=1, workers=4)

# Function to convert sentences to Word2Vec embeddings
def get_word2vec_embeddings(sentences, model):
    embeddings = []
    for sentence in sentences:
        sentence_embedding = [model.wv[word] for word in sentence if word in␣
 ↪model.wv]
        if sentence_embedding:
            embeddings.append(np.mean(sentence_embedding, axis=0))
        else:
            embeddings.append(np.zeros(model.vector_size))  # Use zero vector␣
 ↪for out-of-vocabulary words
    return np.array(embeddings)

# Get Word2Vec embeddings for training and test sets
X_train_word2vec = get_word2vec_embeddings(X_train_tokenized, word2vec_model)
X_test_word2vec = get_word2vec_embeddings(X_test_tokenized, word2vec_model)
```

[46]:

```
print(X_train_word2vec[0])
print(X_train_word2vec.shape[0])
```

```
[-0.01507651  0.01762615  0.00893794  0.00571378  0.00771352 -0.03804718
  0.00651355  0.06112844 -0.01745306 -0.02009552 -0.01349582 -0.04309324
```

```
 -0.00679654 -0.00227248  0.01198575 -0.01461344  0.0089156  -0.02911423
 -0.00325201 -0.05391282  0.01123997  0.01665099  0.0185834  -0.02037119
 -0.00851558  0.00380855 -0.02415785 -0.00945665 -0.02054837  0.00607358
  0.0268063   0.00900034  0.01204579 -0.00988821 -0.01807816  0.03013154
 -0.00057655 -0.02464344 -0.01832067 -0.0482339   0.00653539 -0.02331532
 -0.00403464 -0.00221772  0.02382098 -0.0107545  -0.01932116 -0.00292049
  0.01576255  0.01725384  0.01525865 -0.02236766  0.00063382  0.00455134
 -0.01110479  0.01400781  0.00539879 -0.00650528 -0.03408759  0.00862363
  0.00531588  0.00370765 -0.0041508  -0.0106286  -0.03356361  0.03072255
  0.01505032  0.02049618 -0.03555367  0.03398346 -0.01640854  0.01005498
  0.02386099 -0.0027424   0.01997182  0.00731544 -0.00657559 -0.01022564
 -0.02611655  0.01439636 -0.01167726 -0.00202525 -0.02070466  0.04228414
 -0.00572458 -0.00537303  0.00158786  0.03787795  0.03117262  0.00605081
  0.0265788   0.01497545  0.00341067  0.0042084   0.04504745  0.03313354
  0.02072175 -0.01892107  0.01141014  0.00339946]
781
```

[44]:

```python
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score,confusion_matrix

gnb = GaussianNB()

gnb.fit(X_train_word2vec,y_train)

y_pred = gnb.predict(X_test_word2vec)

accuracy_score(y_test,y_pred)
```

[44]:

```
0.045918367346938778
```

[49]:

```python
from sklearn.tree import DecisionTreeClassifier

clf = DecisionTreeClassifier(random_state=42)

clf.fit(X_train_word2vec,y_train)

y_pred = clf.predict(X_test_word2vec)

accuracy_score(y_test,y_pred)
```

[49]:

```
0.30612244897959184
```

[50]:

```python
rf = RandomForestClassifier(random_state=42)

rf.fit(X_train_word2vec,y_train)
y_pred = rf.predict(X_test_word2vec)

accuracy_score(y_test,y_pred)
```

[50]: 0.4897959183673469

[51]:
```
from sklearn.svm import SVC
from sklearn.metrics import classification_report

svm_classifier = SVC(kernel='linear', decision_function_shape='ovr')

svm_classifier.fit(X_train_word2vec,y_train)

y_pred = svm_classifier.predict(X_test_word2vec)

accuracy_score(y_test,y_pred)
```

[51]: 0.32142857142857145

[52]:
```
from sklearn.neighbors import KNeighborsClassifier

knn_classifier = KNeighborsClassifier(n_neighbors=13)

knn_classifier.fit(X_train_word2vec,y_train)

y_pred = knn_classifier.predict(X_test_word2vec)

accuracy_score(y_test,y_pred)
```

[52]: 0.4387755102040816

[53]:
```
import xgboost as xgb

dtrain = xgb.DMatrix(X_train_word2vec, label=y_train)
dtest = xgb.DMatrix(X_test_word2vec, label=y_test)

params = {
    'objective': 'multi:softmax',
    'num_class': 14,
    'eval_metric': 'merror'
}

num_rounds = 100
xgb_model = xgb.train(params, dtrain, num_rounds)

y_pred = xgb_model.predict(dtest)

accuracy_score(y_test,y_pred)
```

[53]: 0.5408163265306123